TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 18, No. 5, October 2020, pp. 2412~2420
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v18i5.14305

Database techniques for resilient network monitoring and inspection

Zahraa A. Jaaz [1], Suha Sahib Oleiwi[2], Seba Aziz Sahy[3], Israa Albarazanchi[4]
[1]College of Science Computer department AlNahrain University, Iraq,
[2]Department of Computers, Ministry of Higher Education, Iraq
[3]Ministry of Higher Education and scientific Research, Foundation of Technical Education, Institute of Medical Technology Al-Mansur, Iraq
[4] Baghdad College of Economic Sciences University, Iraq

Article Info

Article history:

Received Oct 10, 2019
Revised Jan 16, 2020
Accepted Feb 23, 2002

Keywords:

Database techniques
Elasticsearch
Monitoring
Network
Query optimization
Write optimized database

ABSTRACT

Network connection logs have long been recognized as integral to proper network security, maintenance, and performance management. This paper provides a development of distributed systems and write optimized databases: However, even a somewhat sizable network will generate large amounts of logs at very high rates. This paper explains why many storage methods are insufficient for providing real-time analysis on sizable datasets and examines database techniques attempt to address this challenge. We argue that sufficient methods include distributing storage, computation, and write optimized datastructures (WOD). Diventi, a project developed by Sandia National Laboratories, is here used to evaluate the potential of WODs to manage large datasets of network connection logs. It can ingest billions of connection logs at rates over 100,000 events per second while allowing most queries to complete in under one second. Storage and computation distribution are then evaluated using Elastic-search, an open-source distributed search and analytics engine. Then, to provide an example application of these databases, we develop a simple analytic which collects statistical information and classifies IP addresses based upon behavior. Finally, we examine the results of running the proposed analytic in real-time upon broconn (now Zeek) flow data collected by Diventi at IEEE/ACM Supercomputing 2019.

Corresponding Author:

Israa Albarazanchi,
Baghdad College of Economic Science University,
Baghdad, Iraq.
Email: israa4444@gmail.com

1. INTRODUCTION

The ability to analyze traffic allows network operators to identify network errors, detect anomalous traffic, classify malware [1], find botnets [2], and more. This analysis can be performed as part of an investigation into a detected breach or as part of a real-time analytic to give network insight or detect zero-day attacks as they occur. In order to enable these applications, the system storing network flows faces many challenges. It must simultaneously store a colossal amount of data, ingest event logs at an extremely high rate, and quickly respond to queries. The importance of many events is not known at the time the associated log is generated therefore a large amount of dfata must be stored. The event could be indicative of a misconfiguration or of malicious activity, but the system may not know immediately. This is especially true in the case of security monitoring. In 2018 a majority of successful breaches were not discovered for weeks (<20%), months (40%)

or years (20%). [3] Therefore, forensic analysis of intrusions could necessitate the long-term storage of these events.

Additionally, data must be ingested at a high rate in order to keep pace with network events. The system must ingest at least one second's worth of network data in one second so that it does not rapidly fall behind. Queries upon stored logs need to complete quickly so that analytics running in real time get actionable results, to enable rapid automated analysis of past events, and to ensure the sanity of network security operators using the system. Many existing methods for storing network flow fail to manage these competing interests:

- Problem statement

In order to improve data ingestion, query performance, and capacity, ability to analyze traffic allows network operators to identify network errors, detect anomalous traffic. Network connection logs have long been recognized as integral to proper network security, maintenance, and performance management. The purpose of presenting the following insufficient storage methods is not to make an argument that distributing data or WODs are the only two solutions to this challenge, but rather to further illustrate its complexity. This work presents its data among many nodes in the cluster. This allows insertions and queries to different shards to run in parallel and additionally allows operators provide the system with more resources without scaling up one device. Scale-up quickly becomes prohibitively expensive, but scale-out is much more cost effective. Data integrity is also ensured by duplicating the data across nodes. If the amount of data stored on each node is not too large, the drawback noted above can now become an advantage to the system as it increases the likelihood that data is found on many shards instead of just one.

- Contributions

The main contributions of this work involve improving the results of previous systems in the domain of networking and database, while we compare Diventi and Elasticsearch here, it is worth noting that they are not mutually exclusive, and both provide unique benefits. As an example, use case of these databases, we created a simple analytic to run through the network history of an IP address. This script produces histograms of the following metrics: number of packets in and out, number of bytes in and out, source and destination port numbers, and number of connections with each neighbor. We utilize these metrics to classify the IP addresses as exhibiting one or more network behaviors. We discuss methods for storing network flows, then compare and contrast Elasticsearch and Diventi in section 1.6. Then, in section 2, current and potential network flow analysis of large flow datasets are discussed. Finally, in section 3, we present the results of analytic we developed, the performance of the analytic running in real-time at SC19 (supercomputing 2019), and the results we gleaned from this information. Elasticsearch [4] and Diventi [5] are two examples of systems which properly address these concerns, yet the means by which they do are very different. Elasticsearch is a "real- time distributed search and analytics engine." which allows for both rapid ingestion and query response by sharding data across many nodes in a cluster. Diventi on the other hand leverages the write optimizations of a B to the epsilon tree (Bε-tree) to keep up with data ingestion needs while utilizing the underlying B-tree structure to ensure timely queries. This allows Diventi to store a high amount of data on a single node.

- Network flow database

A network flow is a unidirectional stream of packets with common source and destination. Netflow and IPFIX, two common flow export protocols, aggregate packets from this stream within a given window of time into a single flow. Bro-conn logs [6] are not truly flows, as each log refers to a single bidirectional connection which may be composed of multiple packets to open the connection, send data, and close the connection. All the packets in this connection are aggregated, and for the purposes of this research paper we will refer to bro-conn logs as flows for the sake of simplicity. Packet aggregation inherently results in the loss of packet specific information [7]. This was an early sacrifice made to address the extremely high volume of events created by logging every packet. However, in spite of this reduction in volume, network flow data still suffers from Big Data complexity. "Big Data is data whose complexity hinders it from being managed, queried and analyzed through traditional data storage architectures, algorithms, and query mechanisms." This complexity is defined by the data's volume - the quantity of data to be stored; variety - the system must simultaneously hold un-structured, semi-structured, and structured data; and velocity - the pace at which data is generated for the enterprise network for NIDS in Figure 1. [8] Thankfully, variety is not of concern as network flows all follow a similar structure. Unfortunately, any system tasked with storing this data will still have to contend with high volume and velocity. This complexity hinders unsophisticated efforts to manage, query, and analyze the data.

The fields of network flows which are commonly used as database keys include the timestamp, originating IP address and responding IP address. This research paper is primarily concerned with queries upon the history of IP addresses and subnets. As such, methods discussed here use IP addresses as keys. While not necessary for these applications, full-text search is certainly an attractive feature as it allows for specific queries searching across multiple fields simultaneously.

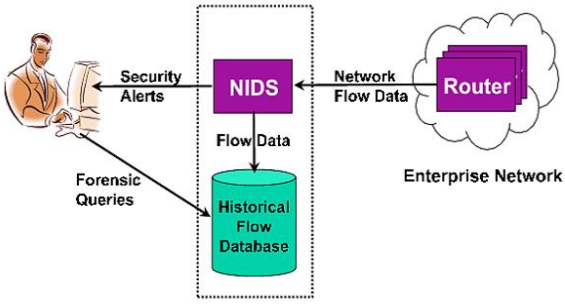

Figure 1. The design for the network intrusion detection using the network flow data and stored historical data in the database as forensic queries were asked by the security analyst for the network monitoring and threat inspection [9]

- Naïve methods

The purpose of presenting the following insufficient storage methods is not to make an argument that distributing data or WODs are the only two solutions to this challenge, but rather to further illustrate its complexity [10]. A first attempt to match the speed of the network could be to directly write each log to a file, however querying for a individual log would require a O(n) search through the database. For real-time analysis and responsive queries this is unacceptably slow, especially as n grows to months or years of data. In response to this setback we might attempt to utilize a data structure such as a hash table to allow a query for a single log to complete in O (1) time. However, this approach fails to take into account volume, as it is only effective while a large portion of the stored data can fit within RAM [11]. This limitation of data to RAM is a direct result of the benefits hashing the key normally provides. The avalanche effect, whereby small changes to the text create a large difference in the hash value, ensures that hashed keys are spread across the used storage space. This is an important feature of hashing allowing the hash table to reduce collisions, but inevitably results in an increasing ratio of disk IOs per inserted log [12]. A large number of disks IOs causes rapid degradation of the insertion rate to below the pace of the network. The only solution would be to keep a majority of the data in RAM, but this limits data retention as increasing the size of RAM becomes prohibitively expensive. NfSen and Flowscan are two common tools used to analyze flows and both suffer from the performance limitations noted above. These methods use a Round Robin Database (RRDtool) [8] [13] to store the data. As a consequence, the IP addresses are unordered, and as discussed above must be kept within RAM in order to facilitate quick searches. RRDtool is a time series database which maintains a constant system footprint by automatically overwriting the oldest values with the newest, once the maximum size of the database is reached [14]. However, this severely limits how long network administrators can store potentially important network flows. Many other potential solutions fall into either of these traps. Elasticsearch and Diventi serve as two examples of how to properly avoid them.

- Elastic search

Elasticsearch uses an inverted index as its underlying data structure to allow data to be stored in the order it arrives while maintaining query performance. An inverted index Figure 2 [4] is a structure which lists all the unique values that appear in any document and the documents in which that value appears. The index can be imagined a map through the unorganized data which allows queries to quickly find what they're looking for. When logging network flows, documents are individual logs and indices include fields such as the originating IP address. Elasticsearch creates an index for each field in the log, allowing operators to query by timestamps, IP addresses, ports, and more.

When preforming a query, the inverted index returns a list of matching documents which can then be retrieved with minimal search cost. However, one drawback of this structure is that for queries on indices other than time, matching documents will be scattered throughout storage. This means that a single node is unable to take advantage of spatial locality and will likely have to contend with a high ratio of memory IOs to access all the documents. In order to improve data ingestion, query performance, and capacity, Elasticsearch shards its data among many nodes in the cluster. This allows insertions and queries to different shards to run in parallel and additionally allows operators provide the system with more resources without scaling up one device. Scale-up quickly becomes prohibitively expensive, but scale-out is much more cost effective. Data integrity is also ensured by duplicating the data across nodes. If the amount of data stored on each node is not too large, the drawback noted above can now become an advantage to the system as it increases the likelihood that data is found on many shards instead of just one.

- Diventi

Write optimized datastructures are designed to provide efficient write performance at the expense of a limited query performance penalty. A B^{ε}-tree is a B-tree with an insertion buffer placed at each node. Data is

inserted to the root buffer which, when filled, flushes its contents to its children Figure 3 [15]. This structure has a few key benefits over a *B*-tree in exchange for paying a small-time penalty on queries. The cost of inserting is $O(\log_B N)$, as opposed to a *B*-tree's cost of $O(\log_B N)$. N is the number of entries in the tree and $B^{1-\varepsilon}$ is the size of the buffer. This may seem to be a small difference in performance, but has much larger implications upon the capacity the database can maintain while keeping pace with the network data. If the system must complete X insertions in one second then the maximum number of logs N is reached when $(\log_B N) X = 1$. $\log N$ grows logarithmically with respect to N; therefore, the maximum value of N increases exponentially with respect to the buffer size [16].

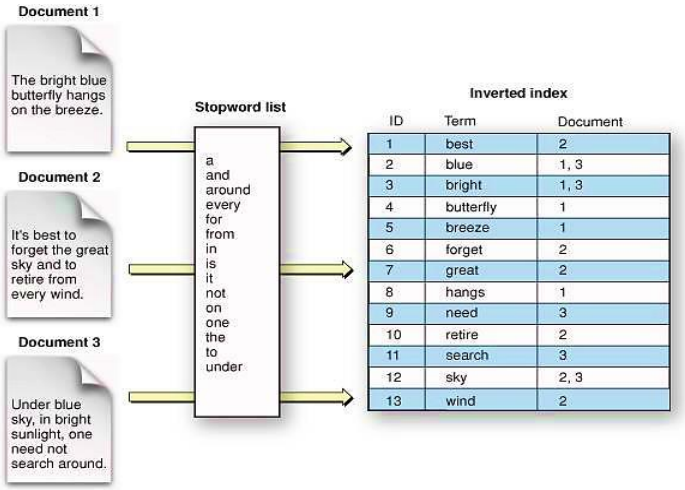

Figure 2. Constructing an inverted index for the database, the index can be imagined a map through the unorganized data which allows queries to quickly find what they're looking for. When logging network flows, documents are individual logs and indices include fields such as the originating IP address [4]

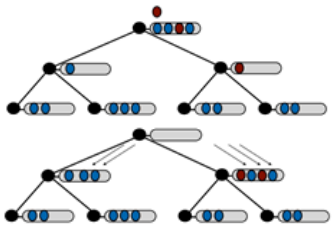

Figure 3. B^{ε}-tree: Insertion of red data item triggers flush to the child nodes, data is inserted to the root buffer which, when filled, flushes its contents to its children that make up a tree [5]

As another performance benefit, writes to disk are amortized because buffers higher in the tree are held within caches and RAM. This means that only those flushes which reach the lower levels of the tree trigger blocking disk IOs. This increases the rate of ingestion. The time penalty to query performance is a consequence of the need to search through the buffer of each node visited while traversing the tree. This means that there is a positive correlation between buffer size and query latency. For more details see Raizes et al. [17]. Diventi orders data first upon the source ip addresses and then the timestamp. The benefit of this is that queries to IP addresses and subnets are quick. The logs which match the query will be contiguous within the database and quick to identify as a result of the B-tree structure. The system can take advantage of spacial locality when performing these queries in addition to quickly identifying the matching logs. As such, disk IO penalties during queries should be minimized. The drawback is that the system can not efficiently query on any other field, however, the workload we are concerned with is primarily investigations into individual ip addresses or subnets. Another drawback is that in order to retrieve logs when the queried IP address matches either the source or responding IP address, two logs of each event must be inserted. One with normal IP ordering and the other reversed. Diventi provides efficient use of resources to allow a large amount of data to be stored on a single node while maintaining high performance. Multiple Diventi nodes responsible for different network taps or for data that is split between them by another process may be deployed if even more capacity is required.

2. METHODOLOGY

Network flows do not contain packet payloads information, and do not provide packet level granularity for fields such as the number of bytes per packet or TCP flags. Instead, flows contain aggregate totals of the number of packets, bytes, flags used in any packet, and more. Despite this loss of specifics, flows still contain sufficient information to identify network intrusions [18]. This section describes existing and potential network flow analysis to be improved or enabled by network flow databases.

2.1. Current methods and research

- Blacklist approach

At the most basic level Nfsen and Flowscan (performance discussed in subsection 1.4) provide for analysis of flow statistics and traffic. They create statistical summaries and graphical displays of data in addition to providing the ability to filter results by a variety of fields. Nfsen additionally provides the ability to define alerts. These alerts can act upon a filtered subset of the overall traffic, trigger on up to 6 chained conditions, and take a set of actions [19]. Similarly, the Bro-IDS provides the ability to trigger actions, such as blocking traffic and creating alerts, based upon the content of the packets4 collected by the tap [20]. This is often accomplished using IP blacklists and checking packet contents against common malware patterns.

- Zero-day attacks

Statistical analysis and machine learn- ing are employed for detecting and classifying a wide variety of network traffic patterns. However, they commonly share the goals of reducing the false positives generated by systems like those discussed above, and detecting zero-day attacks. There are many examples of using network flows to detect and classify actions taken by network participants. Moustafa et al. present an ensemble-based technique for detecting exploits of IoT systems, particularly botnets, using statistical summaries provided by the Bro-IDS [2]. MalClassifier, a tool developed by researchers at Oxford [21], uses the network flow behavior of malware to classify it into various malware families without requiring sandbox execution [22]. MalCalssifier additionally has the ability to determine if the malware does not fit previously established malware families, allowing security operatives to propose new families. Finally, Rodriguez et al. present work in using time series databases studying historical patterns to predict future behavior and detect anomalies. They state that the more data that is used in the time series the more accurate the predictions will be.

2.1.1. Potential application

This portion of the research paper attempts to address the possible uses of a fully operational elasticsearch or diventi database

- Lateral network movement

Lateral network movement is a process by which an attacker takes advantage of access to one machine in the network to gain access to another machine. This is done for the purpose of reconnaissance to find future targets, to reach an objective, or gain a higher level of access to the network. Detecting how a bad actor or piece of malware has moved within one's network is essential for a proper response to an intrusion. Otherwise, malware may remain within the network, continuing to cause damage after action has been taken to address the compromise. Additionally, this type of monitoring may allow security operators to detect anomalies. A chain of ssh logins may be indicative of an attack. In order to identify this movement, it is necessary to hold a large amount of network data [23]. This requirement necessitates the use of a proper network flow database. Additionally, sensors that monitor local traffic are required. The more network visibility the system is given the better, if the system has no view of the connection between computer A and B than it cannot detect lateral movement between them. Tracking lateral movement was considered for this research paper but was ultimately forgone as a result of limited network visibility.

- Machine learning, human interaction and verification

The current work presented in subsection 2.2 is useful in detecting anomalous network behavior and zero-day attacks. However, for complicated use cases, a human security operator will likely have to interact with the machine learning algorithm to verify that the correct actions were taken or to interpret results. To facilitate this, it may become necessary for the user to look into the history of ip addresses which the algorithm has flagged. These queries by the human operator need to complete quickly and have access to a large amount of data in order to facilitate the interaction and save valuable analyst time.

2.1.2. IP flow analysis at supercomputing

To begin to evaluate the performance and use of a network flow database, we developed an analytic to produce flow metrics in real-time at SC19 (supercomputing 2019). Diventi indexed bro-conn logs from a tap used by the SciNet Security Team over the course of the event. At the time the analytic was run, Diventi had indexed a quarter of a billion events corresponding to half a billion logs. What follows is a description and evaluation of the analytic.

2.1.3. Description

The analytic is designed with the goal of gathering basic flow statistics about an IP address. We collected the total number of connections, number of packets in and out, number of bytes in and out, source and destination port numbers, and neighbors. Diventi records the magnitude of the packets and bytes by storing $\log_2 x$ rather than the exact number x. For example, records with byte counts between 1 and 3 are recorded using magnitude 1 and records with a byte count from 4 to 7 recorded with magnitude 2. The number of packets and bytes is therefore already bucketed so as to obscure small differences and highlight large ones [24]. The information was collected in real time and then processed to see what basic conclusions we could reach from the data. We classify each IP address as follows:

- Active if it has more than 100 connections within our network and inactive if it has less than 20.
- High degree (number of neighbors) if the degree is greater than 30 and a small degree if less than 5.
- Receiver if it receives twice the number of packets it sends and a sender if the opposite is true
- Elephant if the average number of bytes sent or received per connection is greater than 10, 000 bytes, a mouse if both are less than 1000 but greater than 80, and a gnat if both are less than 80.
- These categories were defined somewhat arbitrarily with the goal of demonstrating the ability to quickly categorize the behavior of an IP address. As discussed earlier, deep inspection into the history of IPs or subnets is the purpose of this work.

2.1.4. Query performance

We first discovered that the performance of queries across a range of logs was fairly constant even as the size of the database grew much larger. We posit that this is because the majority of a query workload is composed lateral scans through the leaves of the tree. Therefore, increases in the smaller cost to traverse down the tree and find the first matching key are relatively insignificant. Especially as the cost of traversal increases as the log of the number of records. We show this trend in Table 1 [9]. Table 1 shown serverside latency, averaged over 3 queries. Query latency increased by 20% while the size of the database increased by nearly 1000% over the same period, showing the relatively flat latency. In order to ensure uniform results, the query time is an average of 3 queries [9].

1 million logs were queried at intervals when the database had stored between 1 million and 1 billion logs. Query latency increased by 20% while the size of the database increased by nearly 1000% over the same period, showing the relatively flat latency. In order to ensure uniform results, the query time is an average of three queries and the quesy processing and optimization with high level language as shown in Figure 4 [12]. The server was shut down between each query to prevent the results from being cached. A Dell PowerEdge R520 with 165GB of RAM and 32 cores was used for this test, however, the size of the RAM was not a significant factor, as Diventi does not preemptively load data from the underlying storage into its cache.

Table 2 [11] shows the performance of the metrics analytic running upon diventi at supercomputing. Analytic performance for creating metrics end client's end, Real refers to the total time from the start of the program to completion, User to the amount of time the process was executing on the CPU, and System to the amount of time the process was executing system calls [11]. The Unix utility time was used to measure the total amount of time required to create the metrics on the client's end. To establish the performance and generate data we queried a single IP address, a 255.255.255.0 subnet mask, a 255.255.0.0 subnet mask, and the entire database [25]. Real refers to the total time from the start of the program to completion, User to the amount of time the process was executing on the CPU (central processing unit), and System to the amount of time the process was executing system calls. We see an increasing amount time spent off the CPU in Table 2 likely because of the increasing size and complexity of the data stored on the client end, causing IO blocking when preforming analysis. The analytic was able to very quickly establish the statistical history of an ip address or subnet by taking advantage of the Bε-tree's structure. This provides evidence that network flow databases will allow complex analysis to complete rapidly.

Table 1. Serverside latency, averaged over 3 queries

Stored Logs	Query Latency (S)[a]
1	1.947013
100	2.058103
200	2.167957
500	2.208140
1000	2.337618

Table 2. The performance of the metrics analytic running upon diventi at supercomputing

Description	Logs Processed	Real	User	System
Single IP	1,005	0.079	0.024	0.028
Subnet \8	32,864	0.425	0.100	0.076
Subnet \16	372,755	2.386	0.936	0.336
Everything	485,997,746	104 min	22 min	16 min

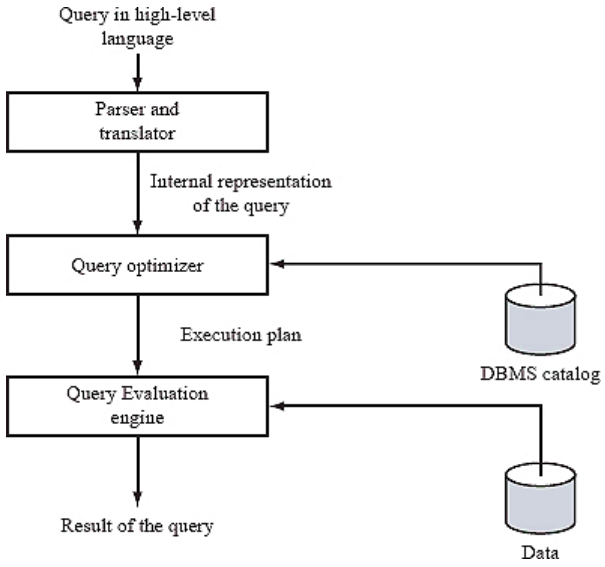

Figure 4. Quesy processing and optimization with high level language query through three levels of query optimization i.e. parser and translator, query optimizer and query evaluation engine to generate the real time results of optimized query by inserting data from the database [12]

3. RESULTS

The result shows graphical representations of the metrics collected upon a single IP address shown in Figures 5-9. This IP had 603 connections, the source port was scattered, but the destination port was always 13568. From this information the IP address is classified as active, of small degree, neither a sender nor receiver, and a mouse. This IP address was likely receiving and sending data to a small number of other IP addresses from a process running on port 13568. The neighbor histogram indicates that the behavior of this IP address is likely most dependent upon neighbor 4.

At the time the analytic was run 1,480,024 IP addresses were stored within the database. Based upon the statistical summary returned when the analytic was run across the entire database each IP address was matched with the classifications described in subsection 2.4. The number of IP addresses that match each category are shown in Table 3 along with the percentage of the total IPs which matched. Using this table, it is observed that a vast majority of IPs were classified as inactive, small degree, senders, or gnats. Based upon this information, we could posit that a vast amount of the traffic collected was composed of simple interactions which didn't require much data to be transferred back to the receiver therefore most packets were sent by the originator. Some examples of this type of traffic include DNS (domain name service) lookups, ICMP (internet control message protocol) messages, and SYN scanning.

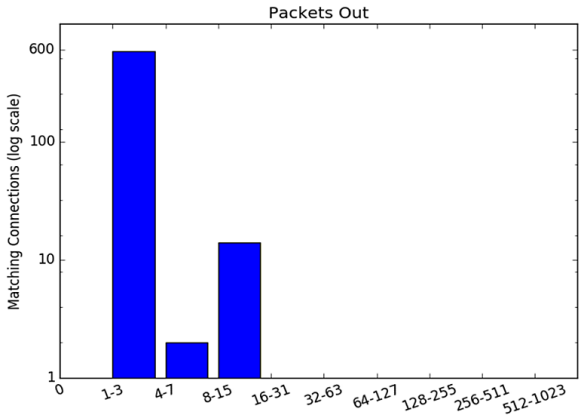

Figure 5. Packet-Out for single IP address on X-axis. Log scale for logging with querying network flows and matching connections on Y-axis.

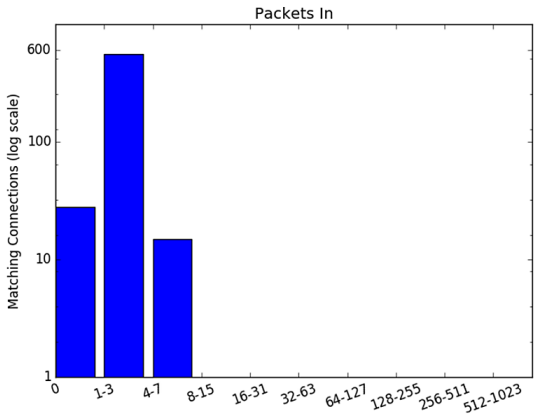

Figure 6. Packet-In for single IP address on X-axis. Log scale for logging with querying network flows and matching connections on Y-axis.

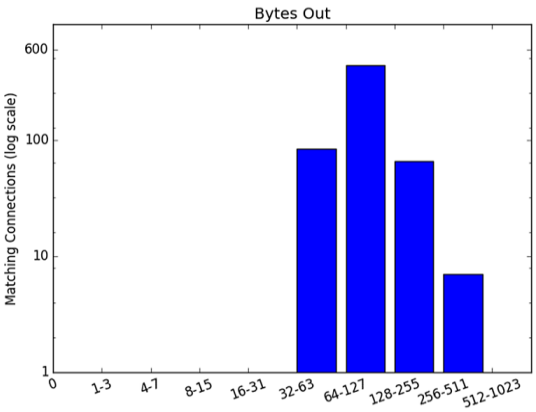

Figure 7. Bytes-In for single IP address on X-axis. Log scale for logging with querying network flows and matching connections on Y-axis.

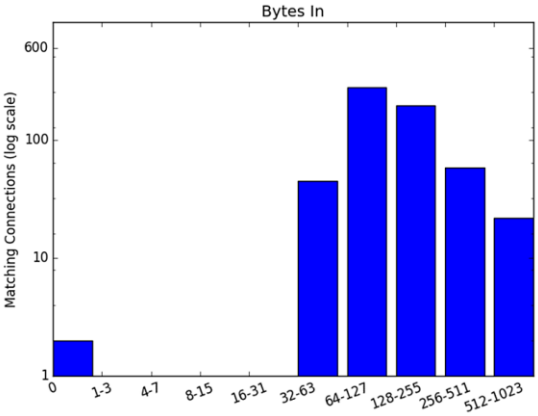

Figure 8. Neighbor histogram with four different neighbors on X-axis. Log scale for logging with querying network flows and matching connections on Y-axis

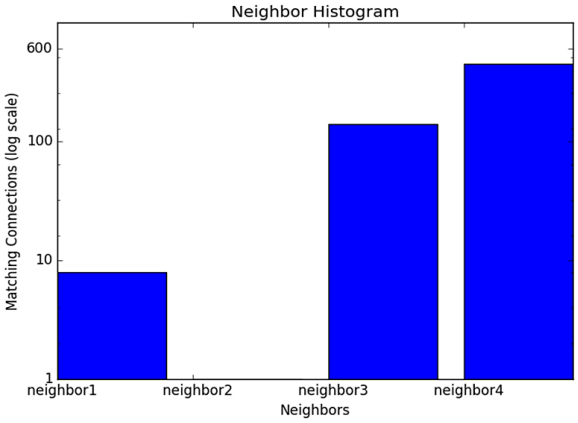

Figure 9. Neighbor histogram with four different neighbors on X-axis. Log scale for logging with querying network flows and matching connections on Y-axis

Table 3. Classifications of supercomputing traffic

Classification	Number of Matches	% of Total
Inactive	1,176,097	79.46
Active	128,218	8.66
Small degree	995,741	67.28
High degree	204,491	13.82
Senders	1,187,979	80.27
Receivers	77,541	5.24
Elephants	14,789	1.00
Mice	94,610	6.40
Gnats	1,348,347	91.10

4. DISCUSSION

First, it should be noted the analytic provides no ability to determine the portion of IPs which matched multiple categories. Each set of classifications: inactive and active, small degree and high degree, etc. is calculated in isolation. A simple improvement to the analytic would be to add this capability allowing the user to zero in on particularly rare behaviors. Diventi was granted only limited access to the supercomputing network. As a consequence of this limited network visibility, some results may be incomplete. It is important that any organization seeking to employ network monitoring carefully consider the implications of the visibility provided by their network taps.

5. CONCLUSIONS

Using databases designed for the big data challenges associated with logging and querying network flows is necessary in order to provide network operators with a larger, more efficient window into the network traffic of both past and present. Its imperative for the development of automated analytics and for effective post-intrusion investigations that these methods are adopted when logging network flows. This research paper shows that Diventi was able to ingest over a billion logs while providing rapid query responses with relatively flat latency. These queries are capable of quickly collecting information regarding IP addresses to classify them into various categories. Running this analysis at supercomputing 2019 revealed that a majority of the traffic collected was composed of simple interactions such as DNS (domain name service) lookups, ICMP (internet control message protocol) messages, and SYN scanning. In conjunction with machine learning and other cutting-edge techniques, these databases allow security personnel to use their time to efficiently identify threats and respond to alerts instead of waiting for information.

REFERENCES

[1] B. A. Alahmadi and I. Martinovic, "MalClassifier: Malware family classification using network flow sequence behaviour," *2018 APWG Symposium on Electronic Crime Research (eCrime)*, no. 1, pp. 1-13, May 2018.

[2] S. Ambrogio *et al.*, "Neuromorphic learning and recognition with one-transistor-one-resistor synapses and bistable metal Oxide RRAM," *IEEE Trans. Electron Devices*, vol. 63, no. 4, pp. 1508-1515, April 2016.

[3] Verizon business ready, "2019 Data Breach Investigations Report," [Online] Available: https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf.

[4] Elasticsearch Reference, "The Definitive Guide," [Online] Available: https://www.elastic.co/guide/en/elasticsearch/guide/master/index.html

[5] N. P. Donoghue, *et al.*, "Tracking network events with write optimized data structures: The design and implementation of TWIAD: The write-optimized IP address database," *Proc.2015 4th Int. Work. Build. Anal. Datasets Gather. Exp. Returns Secur. BADGERS 2015*, pp. 1-7, 2017.

[6] Zeek, "Network Security Monitor," [Online] Available: https://www.zeek.org/.

[7] T. Mahmood and U. Afzal, "Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools," *2013 2nd National Conference on Information Assurance (NCIA)*, pp. 129-134, December 2013.

[8] P. Haag Nfsen, "Netflow sensor," [Online] Available: nfsen.sourceforge.net.

[9] D. Plonka Flowscan, "Network Traffic Flow Visualization and Reporting Tool**"** [Online] Available: www.caida.org/tools/utilities/flowscan/.

[10] T. Oetiker, J. Brutlag, and A. Bogaerdt R, "About RRDtool," [Online] Available: https://oss.oetiker.ch/rrdtool/.

[11] D. Hutchison and J. C. Mitchell, "IP Operations and Management," *9th IEEE Interbational Workshop*, 2008.

[12] F. Maternity *et al.*, "No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title," *Int. Rev. Immunol.*, vol. 66, no. 1, pp. 1-15, 2018.

[13] N. J. Qasim, *et al.*, "Reactive protocols for unified user profiling for anomaly detection in mobile Ad Hoc networks," *Periodicals of Engineering and Natural. Science*, vol. 7, no. 2, pp. 843-852, August 2019.

[14] J. Liu, *et al.*, "Software-defined internet of things for smart urban sensing," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 55-63, September 2015.

[15] O. Salman, *et al.*, "Identity-based authentication scheme for the Internet of Things," *Proc. - IEEE Symp. Comput. Commun.*, pp. 1109-1111, June 2016.

[16] S. Chakrabarty, *et al.*, "Black SDN for the internet of things," *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 190-198, Oct 2015.

[17] T. Theodorou, *et al.*, "A multi-protocol software-defined networking solution for the Internet of Things," *IEEE Commun. Mag.*, vol. 57, no. 10, pp. 42-48, Oct 2019.

[18] A. K. Tran, *et al.*, "SDN controller placement in IoT networks: An optimized submodularity-based approach," *Sensors* (Switzerland), vol. 19, no. 24, pp. 1-12, December 2019.

[19] I. Al Barazanchi, *et al.*, "Innovative technologies of wireless sensor network : The applications of WBAN system and environment," *Sustain. Eng. Innov.*, vol. 1, no. 2, pp. 98-105, 2020.

[20] K. Sood, *et al.*, "Software-defined wireless networking opportunities and challenges for Internet of Things: A review," *IEEE Internet Things Journal.*, vol. 3, no. 4, pp. 453-463, August 2016.

[21] Z. Qin, *et al.*, "A software defined networking architecture for the Internet of Things," *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014.

[22] I. Sosa, *et al.*, "Tackling graphical natural language processing's problems with recurrent neural networks," *J. Southwest Jiaotong Univ.*, vol. 54, no. 5, pp. 1-9, Oct 2019.

[23] A. M. Zarca, *et al.*, "Enabling virtual AAA management in SDN-based IoT networks," *Sensors (Switzerland)*, vol. 19, no. 2, pp. 1-24, January 2019.

[24] A. C. Rodriguez and M. R. De Los Mozos, "Improving network security through traffic log anomaly detection using time series analysis," *Computattional Intelligence in Security for Information Systems*, vol. 85, pp. 125-133, 2010.

[25] H. R. Abdulshaheed, *et al.*, "A review on smart solutions based-on cloud computing and wireless sensing," *Int. J. Pure Appl. Math.*, vol. 119, no. 18, pp. 461-486, 2018.